Personal Bookmark System: Test Plan Document

Group:

The Bookmark Express Triumvirate (BET Inc.)

Contributions by:

Nikitas Marangos

Sean Hanrahan

John Wu

Adib Contractor

Jon Eisenstein

Editor:

Nikitas Marangos

+ expected o/p

Bus testing

5

Rutgers University – Department of Computer Science

01:198:431:01 – Software Engineering

Instructor: Alex Borgida

TA: Rashmi Manjunath

November 20, 2005

Table of Contents

Table of Contents....2
Introduction....3
Unit Testing....3
Integration Testing....3
Functional/Acceptance Testing....3
Integration Schedule....4
Testing Responsibilities....5

Testing Levels

Level 1....6
Level 2....9
Level 3....9
Level 4....11
Level 5....11
Level 6....13
Level 7....14
Level 8....16
Level 9....16
Level 10....18
Level 11....19
Level 12....21
Level 13....24
Level 14....25
Level 15....26
Acknowledgements....28

Introduction

This test plan is intended to help testers of the Personal Bookmark System (PBS) find errors in each of the PBS classes. Those errors shown should give some indication as to what the cause of them might be. With each of the 15 levels of the program, the units in each level should be tested individually for errors. Thanks to the program's design, in the process of testing each unit, the integration of the levels it depends on will be tested simultaneously. Successful tests are constituted by being able to purposefully cause errors in a function.

Unit Testing

Each class or class portion at each level will undergo unit testing through JUnit test cases built for it. These test cases will consist of black box testing, involving data picked without awareness of the specific behavior of the class, and white box testing, involving data picked with awareness of the specifc behavior of the class. When a class is initially tested with JUnit, the cases in which errors occur will be recorded and they will be reported by the tester of the class to the implementer of the class. The tester will then give his input as to what the problem might be in each case. With this second opinion and his own hypothesis as to what caused each error, the implementer will try to fix this problem. With each successive fix of the class, all the JUnit test runs for the class will be run again and the process will repeat until practically all possible and likely errors for the class have been corrected.

Integration Testing

Each class or class portion at each level will probably depend on classes and class portions from lower levels. The design of this program allows for integration testing to be done as unit testing at the higher levels. The higher levels use the functions of the lower levels in their functions. At the top level of the program, the GUI, the entire program can be tested.

Functional/Acceptance Testing

After all the integration tests are completed, we will consult the user to see if his requirements for the program have been met. These requirements are largely specified in the requirements document.

Integration Schedule

It is not certain how quickly certain classes will be completely written and tested, but we do know that the following deadlines are mandatory. It's entirely likely that levels will be written and tested before these deadlines, but these deadlines are absolute for our purposes:

Level to be Completed	Date to be Completed By
Level 1	11/28/05
Level 2	11/29/05
Level 3	11/30/05
Level 4	12/01/05
Level 5	12/02/05
Level 6	12/03/05
Level 7	12/04/05
Level 8	12/05/05
Level 9	12/06/05
Level 10	12/07/05
Level 11	12/08/05
Level 12	12/09/05
Level 13	12/10/05
Level 14	12/11/05
Level 15	12/12/05

Testing Responsiblities

Nikitas Marangos

Levels 2 - 5

Sean Hanrahan

Levels 1, 8, 9

John Wu

Levels 6, 7

Adib Contractor

Levels 14 - 15

Jon Eisenstein

Levels 10 - 13

Level 1

PBSConfig

It is very important that a user be able to customize his PBS experience. We will offer several options for him to change things to his liking. These options will comprise his configuration or PBSConfig in our case. Testing of this class is done at this level mainly because the PBSPersistence class in Level 9 must be able to build a PBSConfig out of a file.

Test Cases

Case	Method	Input	Expected Output
1	constructor()		creates blank PBSConfig object
2	setOption(name, switch, setting)	name is unique, setting is valid	returns true
3	setOption(name, switch, setting)	name is not unique (is name of another option)	Throws NameTakenException
4	getOption(name)	name is the name of one of the option in the PBSConfig	Return setting for option with name
5	getOption(name)	name is not the name of any option in the PBSConfig	Throws OptionNotFoundException
6	enable(name)	name is the name of one of the option in the PBSConfig	set switch of option with name to true
7	enable(name)	name is not the name of any option in the PBSConfig	Throws OptionNotFoundException
8	disable(name)	name is the name of one of the option in the PBSConfig	set switch of option with name to false
9	disable(name)	name is not the name of any option in the PBSConfig	Throws OptionNotFoundException
10	isEnabled(name)	name is the name of one of the option in the PBSConfig	returns switch of option with name
11	isEnabled(name)	name is not the name of any option in the PBSConfig	Throws OptionNotFoundException

PBSFile

Before PBS can retrieve library backups from the server, it must be certain that it can read from the files whatever it needs. PBS must also be able to read PBSConfigs from files. Thus, we test the basic file operations for PBS here at this level. We assume here that the user has a bookmarks.html file in the ~/.mozilla directory and also that he has a file "stringtest.pbs" containing the serialized string "Hi there".

Test Cases

Case	Method	Input	Expected Output
1	constructor		creates blank PBSFile object
2	fileExists(name, path)	file with name "bookmarks.html" at "~/.mozilla" path	Returns true
3	fileExists(name, path)	file with name and path exists	returns true
4	fileExists(name, path)	file with name does not exist at the path specified	returns false
5	readFile(name, path)	file "bookmarks.html" at "~/.mozilla"	Returns contents of ~/.mozilla/bookmarks.html
6	readFile(name, path)	file with name exists at path	Returns contents of the file at that path
7	readFile(name, path)	file with name doesn't exist at path	throws NonexistentFileException
8	readFile(name, path)	file with name at path isn't readable (ex. may be in a root directory)	throws UnreadableFileException
9	renameFile(name, path, newname)	user wants to rename "bookmarks.html" in his "~/.mozilla" directory to "bookmarksold.html"	Returns true, renaming bookmarks.html to bookmarksold.html
10	renameFile(name, path, newname)	file called name at path exists and can be changed	Returns true, renaming file called name to newname
11	renameFile(name, path, newname)	a user's "bookmarks.html" file or another file is unchangeable	throws UnchangeableNameException
12	writeFile(contents, path, name)	File does not already exist at path with name	Write contents to file with name at path and return true
13	writeFile(contents, path, name)	contents are null or "" and file doesn't already exist	Return true, there is an empty file at path with name
14	writeFile(contents, path, name)	File at path with name already exists (ex. "~/.mozilla/bookmarks.html")	Throws UnwritableFileException
15	readObject(name, path)	file is "~/stringtest.pbs" containing a serialized string	Returns the string that was serialized (ex. "Hi there")

Case	Method	Input	Expected Output
16	readObject(name, path)	File with name at path doesn't exist	Throws NonexistentFileException
17	readObject(name, path)	File at path with name is unreadable	Throws UnreadableFileException
18	writeObject(object, name, path)	object is valid and file at path with name doesn't already exist	Return true, object is written to file
19	writeObject(object, name, path)	object is valid but file at path with name already exists	Throws UnwritableFileException
20	writeObject(object, name, path)	object is null and file at path with name doesn't already exist	Write nothing to file and return true

PBSBookmark

The most basic unit of a user's library is the bookmark. The PBSBookmark seeks to provide a reliable representation of a user's bookmark. It is tested at this level.

Test Cases

Case	Method	Input	Expected Output
1	constructor		blank PBSBookmark object
2	constructor(name, url, date, adddata)	name, url, date, adddata all valid	creates a PBSBookmark with name, address URL, date, and additional data
3	equals (PBSBookmark other)	other is this	return true
4	equals (PBSBookmark other)	other is null	return false
5	equals (PBSBookmark other)	other has same name and URL as current bookmark	return true
6	equals (PBSBookmark other)	other bookmark does not have same name and URL	return false
7	getName()	PBSBookmark created with name “google”	Return “google”

Case	Method	Input	Expected Output
8	getURL()	PBSBookmark created with address "www.google.com"	Returns "www.google.com"
9	getDate()	PBSBookmark created with date D	Returns D
10	getAddData()	PBSBookmark created with additional data A	Returns A
11	toString()	PBSBookmark created with name "google" and URL "www.google.com"	Returns "google www.google.com"

Level 2

PBSNode (PBSBookmark only)

Since PBSBookmark has been tested in Level 1, we can assume that PBSBookmark is valid. We want to have a preliminary set of functions for PBSNode working (those pertaining to PBSBookmarks) so that we can test some functions of PBSFolder (contains a Vector of PBSNodes). These PBSFolders will also be contained in PBSNodes which we will test in Level 4.

Test Cases

Case	Method	Input	Expected Output
1	constructor (PBSBookmark)	valid PBSBookmark	Creates a PBSNode wrapper around the PBSBookmark
2	isFolder()	PBSNode wraps a PBSBookmark p	Returns false
3	getBM()	PBSNode wraps a PBSBookmark p	Returns p
4	getName()	PBSNode was created with name NAME	Returns NAME
5	toString()	PBSNode was created with name NAME	Returns NAME

Level 3

PBSFolder (no subfolders)

Up to this level, PBSNodes containing PBSBookmarks have been tested. In order to test PBSNode completely (with the functions pertaining to PBSFolders) in Level 4, some testing for PBSFolder is in order. However, since we have not yet tested PBSNode for PBSFolders, we cannot test PBSFolder for folders with subfolders at this level. The complete testing of PBSFolder (with subfolders) is done at Level 5.

Test Cases

Case	Method	Input	Expected Output
1	constructor (name, parent)	name is "" or null, parent is null	throws IllegalNameException
2	constructor (name, parent)	name is only whitespace, parent is null	throws IllegalNameException
3	constructor (name, parent)	name is >=1 character and is not only whitespace, parent is null	Creates a PBSFolder with the name
4	add(node)	node is null	throws NullNodeException
5	add(node)	node is for a PBSBookmark	Add node to folder
6	hasNext()	nodeList.size - 1 > current	Returns true
7	hasNext()	nodeList.size - 1 <= current	Returns false
8	next()	hasNext() is true	Increments current, sets calledNext to true and returns next node
9	next()	hasNext() is false	throws NoSuchElementException
10	remove()	calledNext is true (current bookmark has been seen)	Removes the bookmark, decrements counter, and sets calledNext to false
11	remove()	calledNext is false (current bookmark has not yet been seen)	throws IllegalStateException
12	reset()		Sets current back to -1 (right before first bookmark if any in the folder)
13	compareTo (PBSFolder other)	Doesn't have simple test cases. Currently unsure how exactly this function will behave.	Bookmarks that are in both folders will be in an intersection folder. Bookmarks which are not in both folders will be in a disjoint folder.
14	equals(other)	other is this	Return true
15	equals(other)	other is null	Return false
16	equals(other)	Name of this folder is the same as name of other folder	Return true
17	equals(other)	Name of this folder is different from name of other folder	Return false
18	getName()	Folder was created with name NAME	Return NAME

Case	Method	Input	Expected Output
19	toString()	Folder was created with name NAME	Return NAME

Level 4

PBSNode (complete)

At this point, we have tested a PBSBookmark and a PBSFolder that can only contain PBSBookmarks. We can now assume that they are both valid. Since the PBSNode class doesn't particularly care for the contents of the PBSFolder and just needs it to be valid, we will test the full PBSNode class here.

Test Cases

Case	Method	Input	Expected Output
1	constructor(PBSBookmark)	Valid PBSBookmark b	Creates a PBSNode wrapping b
2	constructor(PBSFolder)	Valid PBSFolder f	Creates a PBSNode wrapping f
3	isFolder()	Node wraps a PBSBookmark	Return false
4	isFolder()	Node wraps a PBSFolder	Return true
5	getBM()	isFolder() is false	Return PBSBookmark in node
6	getBM()	isFolder() is true	Throws NotBookmarkException
7	getFolder()	isFolder() is true	Return PBSFolder in node
8	getFolder()	isFolder() is false	Throws NotFolderException
9	getName()	PBSNode created with name NAME	Returns NAME
10	toString()	PBSNode created with name NAME	Returns NAME

Level 5

PBSFolder (with subfolders)

Up to this level, we have fully tested PBSNodes containing PBSBookmarks and PBSNodes containing PBSFolders. We can now fully test PBSFolder, with subfolders and bookmarks. The root PBSFolder containing all bookmarks from a browser will be in a user's PBSVolume, tested in the next level.

Test Cases

Case	Method	Input	Expected Output
1	constructor(name, parent)	name is "" or null, parent is null	throw IllegalNameException
2	constructor(name, parent)	name is only whitespace, parent is null	throw IllegalNameException
3	constructor(name, parent)	name is >=1 character and is not only whitespace, parent is null	Creates a PBSFolder with the name
4	constructor(name, parent)	name is "" or null, parent is not null	throw IllegalNameException
5	constructor(name, parent)	name is only whitespace, parent is not null	throw IllegalNameException
6	constructor(name, parent)	name is >=1 character and is not only whitespace, parent is P (not null)	Creates a PBSFolder with name and with parent set to P
7	add(node)	node is null	throw NullNodeException
8	add(node)	node is a PBSNode wrapping this	throw ThisNodeException
9	add(node)	node wraps a valid PBSBookmark	Add node to folder
10	add(node)	node wraps a valid PBSFolder	Add node to folder
11	hasNext()	nodeList.size - 1 > current	Return true
12	hasNext()	nodeList.size – 1 <= current	Return false
13	next()	hasNext() is true	Increments current, sets calledNext to true and returns next node
14	next()	hasNext() is false	throws noSuchElementException
15	remove()	calledNext is true (current node has been seen)	Removes the bookmark, decrements counter, and sets calledNext to false
16	remove()	calledNext is false (current node has not yet been seen)	throws IllegalStateException
17	reset()		sets current back to -1 (right before first bookmark if any)

Case	Method	Input	Expected Output
18	compareTo (PBSFolder other)	Doesn't have simple test cases. Currently unsure how exactly this function will behave.	Bookmarks and subfolders that are in both folders will be in an intersection folder. Subfolders which are not in both folders will be in a prompt folder (the user will be prompted about these later). Bookmarks which are not in both folders will be in a disjoint folder.
19	equals(PBSFolder other)	other is this	Return true
20	equals(PBSFolder other)	other is null	Return false
21	equals(PBSFolder other)	name of this folder is same as name of other folder	Return true
22	equals (PBSFolder other)	name of this folder is different from name of other folder	Return false
23	getName()	Folder was created with name NAME	Return NAME
24	toString()	Folder was created with name NAME	Return NAME

Level 6

PBSVolume

The PBSVolume represents all of a user's bookmarks from a particular browser. It holds the root PBSFolder which contains the user's bookmarks from a browser. We can assume this root folder is valid because it was tested for reliability in the previous level. It is part of a PBSCollection, which is tested in level 7.

Test Cases

Case	Method	Input	Expected Output
1	constructor (root, browser, name)	root is root folder of volume, volume is associated with browser, and the volume has a name	Creates a new PBSVolume

Case	Method	Input	Expected Output
2	toString()	PBSVolume was created with name N	Returns N
3	getBrowser()	PBSVolume was created for browser named B	Returns B
4	getRoot()	PBSVolume was created with root folder R containing bookmarks and subfolders of PBSVolume	Returns R
5	compareTo (PBSVolume other)		Returns new PBSVolume with root PBSFolder containing disjoint and intersection of two PBSVolumes

Level 7

PBSCollection

The PBSCollection represents all the bookmarks a user has for a particular purpose or location (home, work, school, etc.) Each browser the user uses has its own PBSVolume of bookmarks which we can assume is valid from the testing of PBSVolume in level 6. Because this PBSCollection represents one purpose for the user, it is part of a PBSLibrary which is tested in the next level.

Test Cases

Case	Method	Input	Expected Output
1	constructor(name)		Creates a new PBSCollection with name
2	add(volume)	volume successfully added to collection	Return true
3	add(volume)	volume not successfully added to collection	return false
4	delete(volumename)	volume with volumename deleted successfully from collection	return true

Case	Method	Input	Expected Output
5	delete(volumename)	volume with volumename not successfully deleted from collection (may not exist, etc.)	return false
6	find(volumename)	volume with volumename found in collection	return the PBSVolume
7	find(volumename)	volume with volumename not found in collection	return null
8	remove()		The active PBSVolume is removed from the PBSCollection
9	hasNext()	there is at least one more PBSVolume in the PBSCollection relative to the current active PBSVolume	Return true
10	hasNext()	there aren't any more PBSVolumes after active PBSVolume	return false
11	next()	hasNext() is true	return next PBSVolume
12	next()	hasNext() is false	throws noSuchElementException
13	reset()		internal active volume point is reset to point right before the first volume of the collection

PBSBrowser

The PBSBrowser class will provide PBS with the user's bookmarks on his local machine. These extracted bookmarks will comprise a user's PBSCollection as each browser's set of bookmarks will go into a PBSVolume. The PBSLibrary's bookmarks have to come from somewhere and the user must be able to see the changes in his bookmark collection reflected in the bookmarks on his browser. This class is tested here in level 7 before PBSLibrary:

Test Cases

Case	Method	Input	Expected Output
1	importBookmarks(bname, fname)	String bname which is the name of the web browser and String fname which is the name of the browser?s bookmark file we want to parse.	Returns a new PBS Volume object that contains the bookmarks and folders of the bookmark file taken as input.

Case	Method	Input	Expected Output
2	pushBookmarks (PBSVolume finalVol, String browser)	PBSVolume finalVol which is a volume that contains the final arrangement of bookmarks and folders as the user order them and String browser which is the name of the browser that the file will be exported to.	Returns true if the bookmark file was successfully exported to the browser, false otherwise.

Level 8

PBSLibrary

The PBSLibrary class is tested for its reliability as a representation of all of a user's bookmarks for every purpose. These purposes have their own valid PBSCollections (tested in level 7).

Test Cases

Case	Method	Input	Expected Output
1	constructor		Creates an empty PBSLibrary
2	add(collection)	collection is a valid PBSCollection	Return true
3	remove()		Remove active collection
4	hasNext()	collections.size() -1 > activeCollection	Return true
5	hasNext()	collections.size() -1 <= activeCollection	Return false
6	next()	hasNext() is true	Returns next PBSVolume
7	next()	hasNext() is false	throws noSuchElementException
8	reset()		Reset activeCollection to right before first collection
9	switchCollection(name)	name is the name of a collection in the PBSLibrary	Returns true, making the collection with the name the activ one
10	switchCollection(name)	name is not the name of a collection in the PBSLibrary	Returns false, indicating the collection was not found
11	getActive()		Returns the active PBSCollection

Level 9

PBSPersistence

The PBSPersistence class is responsible for maintaining the reliability of PBSLibraries and PBSConfigs in files. It uses the PBSFile class to accomplish this. Any class which wants a PBSConfig or a PBSLibrary from a file must use this class, so it must be tested here:

Test Cases

Case	Method	Input	Expected Output
1	constructor		creates a new PBSPersistence object
2	getConfig(name, path)	file with name at path contains a PBSConfig object (ex. "~/pbs/config.pbs")	Returns the PBSConfig object
3	getConfig(name, path)	file with name at path unreadable	throw UnreadableFileException
4	getConfig(name, path)	file with name at path nonexistent	throw NonexistentFileException
5	getConfig(name, path)	Object in file not a serialized PBSConfig	throw UnreadableConfigException
6	getLibrary(name, path)	file with name at path contains a PBSLibrary object (ex. "~/pbs/config.pbs")	Returns the PBSLibrary object
7	getLibrary(name, path)	file with name at path unreadable	throw UnreadableFileException
8	getLibrary(name, path)	file with name at path nonexistent	throw NonexistentFileException
9	getLibrary(name, path)	Object in file not a serialized PBSLibrary	throw UnreadableLibraryException
10	writeConfig (config, name, path)	File at name and path doesn't already exist (eg. "~/pbs/confignew.pbs")	Write config to file and return true
11	writeConfig (config, name, path)	File at name and path already exists (eg. "~/pbs/config.pbs")	throw UnwritableFileException
12	writeLibrary(lib, name, path)	File at name and path doesn't already exist (eg. "~/pbs/librarynew.pbs")	Write library to file and return true

Case	Method	Input	Expected Output
13	writeLibrary(lib, name, path)	File at name and path already exists (eg. "~/pbs/library.pbs")	throw UnwritableFileException
14	XMLtoLibrary	XML StringBuffer	Returns PBSLibrary containing contents specified in XML file
15	XMLtoLibrary	XML StringBuffer not in valid PBS Format	throw BadXMLException

Level 10

PBSServer (protocol, no file transfer)

The PBSServer is responsible for storing, sending, and receiving a user's PBSLibraries or PBSConfig. It must first be tested that the protocol used below can allow communication with the BET server and awareness of library backups on the server.
PBSServer is tested with 2 accounts. Account 'testuser' is an account with multiple backups on the server, while 'emptyuser' has none. The passwords for the two accounts are the same. At this level, we are testing the protocol used by the PBSServer and PBSNet classes to send PBSLibraries and PBSConfigs to and receive from the server. To get the password for an account, "USER OK" must first be received to indicate that the named user has an account. We'll test these commands with a dummy sender and a dummy receiver.

Test Cases

Case	Command	Input	Expected Response
1	USER <username>	username is 'testuser'	USER OK
2	USER <username>	username is 'emptyuser'	USER OK
3	USER <username>	username is invalid	USER FAILED
4	USER <username>	username is ''	USER FAILED
5	USER <username>	username is not a user who has an account	USER FAILED
6	QUIT		QUIT OK
7	PASS <password>	received USER OK, password is correct password for account	PASS OK
8	PASS <password>	received USER OK, password is invalid	PASS FAILED
9	PASS <password>	received USER OK, password is ''	PASS FAILED
10	PASS <password>	received USER OK, password is not correct password for account	PASS FAILED

Case	Command	Input	Expected Response
11	LIST	received PASS OK, account has backups on server	LIST START (then names of backups) LIST OK
12	LIST	received PASS OK, account does not have backups on server	LIST START LIST OK
13	LATEST	received PASS OK, account has backups on server	LATEST (name of latest backup)
14	LATEST	received PASS OK, account has 1 backup on server named X	LATEST X
15	LATEST	received PASS OK, account does not have backups on server	LATEST NONE
16	any other command		COMMAND UNSUPPORTED

Level 11

PBSNet (part)

The PBSNet class is the PBS connection to the BET server. It must be certain that the PBSNet class can use the specified protocol in Level 10 to connect to the BET server and become aware of backups that may be on the server. Here it will be made certain that PBS and the BET server can communicate with one another.

Test Cases

Case	Method	Input	Expected Output
	constructor		Creates an instance of PBSNet
	send(socket, cmd)	socket created by connectToBET; connection to BET server drops	throws ConnectionFailedException
	send(socket, cmd)	socket created by connectToBET; connection persists	sends command
	getResponse(socket)	soekct created by connectToBET; connection to BET server drops	throws ConnectionFailedException
	getResponse(socket)	socket created by connectToBET; connection persists	returns command
	sendUserPass(user, pass, socket)	socket created by connectToBET; connection to BET server drops	throw ConnectionFailedException

Case	Method	Input	Expected Output
	sendUserPass(user, pass, socket)	socket created by connectToBET; user is invalid	Return false
	sendUserPass(user, pass, socket)	socket created by connectToBET; password is incorrect for user on server	Return false
	sendUserPass(user, pass, socket)	socket created by connectToBET; user and password are correct	Return true
1	connectToBET(server, port, user, pass)	server is BET server, port is correct, user and pass are correct for an account on server	Return a socket
	connectToBET(server, port, user, pass)	server is BET server, port is correct, user does not exist on server	Throw LoginException
	connectToBET(server, port, user, pass)	server is BET server, port is correct, password is incorrect for user on server	Throw LoginException
	connectToBET(server, port, user, pass)	server is BET server, port is correct, user is ''	Throw LoginException
	connectToBET(server, port, user, pass)	server is BET server, port is correct, user is correct, password is ''	Throw LoginException
	connectToBET(server, port, user, pass)	server is not BET server	Throw ConnectionFailedException
	connectToBET(server, port, user, pass)	server is BET server, port is invalid	Throw ConnectionFailedException
	connectToBET(server, port, user, pass)	server is BET server, port is correct, user and pass are correct for an account on server, connection drops	Throw ConnectionFailedException
	closeBET(socket)	socket created by connectToBET, receives QUIT FAILED when sending a QUIT	Throw QuitFailedException
	closeBET(socket)	socket created by connectToBET, receives QUIT OK when sending a QUIT	Closes socket
	closeBET(socket)	socket created by connectToBET, connection drops	throw ConnectionFailedException

Case	Method	Input	Expected Output
	getLibraryList(server, port, user, pass)	can connect to BET with server, port, user, pass	Returns Vector with list of backup names (if any)
	getLibraryList(server, port, user, pass)	cannot connect to BET with server, port, user, pass or connection is dropped	Throw ConnectionFailedException
	getLatestLibrary (server, port, user, pass)	can connect to BET with server, port, user, pass, account has at least 1 library backup on server	Returns name of latest backup
	getLatestLibrary (server, port, user, pass)	can connect to BET with server, port, user, pass, account has no library backups on server	Returns ""
	getLatestLibrary (server, port, user, pass)	cannot connect to BET with server, port, user, pass or connection is dropped	Throw ConnectionFailedException

Level 12

PBSServer (remaining protocol and functions)

Now we know that PBS and the BET Server can communicate with one another with the protocol specified in Level 10. However, the protocol in level isn't complete. This set of protocol with that in Level 10 will allow PBS to send and receive PBSLibraries and PBSConfig. These protocol will be tested with a dummy server listener and the PBSNet portion from level 11:

Remainder of Protocol

Case	Command	Input	Expected Response
1	GET <datetime>	datetime is a name of a backup which exists on server and is reliable	GET START (file in XML format) GET OK
2	GET <datetime>	datetime is a name of a backup which doens't exist on server or is corrupt	GET FAILED

Case	Command	Input	Expected Response
3	GETOBJECT <datetime>	datetime is a name of a backup which exists on server and is reliable	GETOBJECT STARTs (object sent over bytestream) GETOBJECT OK
4	GETOBJECT <datetime>	datetime is a name of a backup which doesn't exist on server or is corrupt	GETOBJECT FAILED
5	PUTFILE <datetime>	backup named datetime cannot be created on server	PUTFILE FAILED
6	PUTFILE <datetime>	backup named datetime can be created on server and server is ready to receive	PUTFILE OK
7	PUT <data>	received PUTFILE OK; line is not successfully written	PUT FAILED
8	PUT <data>	received PUTFILE OK; line is successfully written	PUT OK
9	PUTOBJECT <data>	received PUTFILE OK; object is not received and written	PUTOBJECT FAILED
10	PUTOBJECT <data>	received PUTFILE OK; object is received and written successfully	PUTOBJECT OK
11	FINISHFILE	received PUTFILE OK; file cannot be written to	FINISHFILE FAILED
12	FINISHFILE	received PUTFILE OK; file named N written to successfully	FINISHFILE N
13any other command			COMMAND UNSUPPORTED

In order for the PBSNet class to do all that it needs to do, it must be certain that a PBSServer can follow the protocol and handle all of its requests. The PBSServer will be tested in its entirety here:

Test Cases

Case	Method	Input	Expected Output
1	constructor(port)	port is an unused and acceptable port number	starts server on specified port

Case	Method	Input	Expected Output
2	runServer()		continuously waits for incoming connections and sends commands these connection send to commandHandler()
3	commandHandler()		waits for commands, sends it to getCmd(command)
4	getCmd(String command)	command is USER <user>	sends PBSServer response to USER command as in level 10
5	getCmd(String command)	command is PASS <pass>	sends PBSServer response to PASS command as in level 10
6	getCmd(String command)	command is LIST; user can connect to BET	sends PBSServer response to LIST command as in level 10 with library list from outputLibraryList()
7	getCmd(String command)	command is LATEST; user can connect to BET	send PBSServer response to LATEST command as in level 10 with latest library name from findLatestLibrary()
8	getCmd(String command)	command is GET <timestamp>; user can connect to BET	sends PBSServer response to GET command as in level 12 with library obtained from outputLibrary (timestamp)
9	getCmd(String command)	command is GETOBJECT <timestamp>; user can connect to BET	sends PBSServer response to GETOBJECT command as in level 12 with library obtained from outputLibraryObj(timestamp)
10	getCmd(String command)	command is PUTFILE <timestamp>	sends PBSServer response to PUTFILE command as in level 12 with library obtained from inputLibrary()
11	getCmd(String command)	command is QUIT	sends PBSServer response to QUIT command as in level 12
12	verifyUserPass (user, pass)	pass is correct for user	return true
13	verifyUserPass (user, pass)	pass is incorrect for user	return false

Case	Method	Input	Expected Output
14	verifyUserPass (user, pass)	user doesn't exist on server	return false
15	getPassword(user)	user doesn't exist on server	throws NoUserException
16	getPassword(user)	user exists on server	return user's password
17	outputLibraryList()		returns a Vector of timestamps for all the library backups on server
18	findLatestLibrary()	user has library backups on server	returns timestamp of last uploaded library
19	findLatestLibrary()	user doesn't have library backups on server	returns 'NONE'
20	validGetFile(time)	file with time is valid	return true
21	validGetFile(time)	file with time is invalid	return false
22	outputLibrary (time)	validGetFile(time) is true	sends lines in library backup line by line
23	outputLibraryObj (time)	validGetFile(time) is true	sends object in file over bitstream
24	validPutFile(time)	file with time is valid	return true
25	validPutFile(time)	file with time is invalid	return false
26	inputLibrary(time)	validPutFile(time) is true	attempts to write to file with time
27	closeConnection()		close socket

Level 13

PBSNet (remainder)

The PBSNet class now has a PBSServer awaiting any reasonable requests it might make. It must now be completed so that PBSParser (the "workhorse" of the program) may use it to get PBSLibraries from the BET server. It must be able to handle the complete protocol between PBSNet and PBSServer and get and send PBSLibraries. Along with the tests done on PBSNet in Level 11 (which MUST be re-run on this level), the following tests must be run:

Test Cases

Case	Method	Input	Expected Output
1	getLibraryObject(server, port, user, pass, library)	server is BET server, port is correct, user and pass match an account, library file is valid	Returns PBSLibrary
2	getLibraryObject(server, port, user, pass, library)	server is BET server, port is correct, user and pass match, library file is invalid	throws GetFailedException
3	uploadLibraryObject (server, port, user, pass, library)	server is BET server, port is correct, user and pass match, library object is valid	Returns timestamp of backup

Level 14

PBSParser

The PBSParser is the main program the user doesn't see. It must be able to send and receive PBSLibraries and PBSConfigs, it must be able to compare PBSCollections, and it must be able to extract the bookmarks from the user's local machine. It must essentially be able to do whatever the user wants it to do through the GUI. It must be tested after all the other tasks are tested but before the GUI. It is tested here:

Test Cases

Case	Method	Input	Expected Output
1	constructor		Creates new objects for all global properties
2	getLibFile()	Connection persists to get library file	Return user's current library
3	getLibFile()	Connection fails during library file retrieval	Returns null
4	loadConfig()	Values from user's config file correctly loaded into PBSConfig	Returns PBSConfig with config values
5	loadConfig()	Values from config file not correctly read	Throw exceptions from PBSPersistence to GUI

Case	Method	Input	Expected Output
6	writeLibrary(library)	valid PBSLibrary library	Write library to local system using passed object as well as config values for library
7	writeLibrary(library)	library is null or invalid	Throws IllegalObjectException
8	parseLocalBrowsers ()	Local browser files are valid	Return PBSCollection containing volumes for each browser.
9	parseLocalBrowsers ()	Local browser files are invalid	Throw exceptions from PBSBrowser to GUI
10	switchCollection (name)	name is a name of a collection	Switch active collection to one specified and return true
11	switchCollection (name)	name is invalid, "", or otherwise not of a collection in library	Throw IllegalNameException
12	compareCollection (PBSCollection left, PBSCollection right)	left and right are PBSCollections with volumes that have some bookmarks and folders in common	Returns a collection containing a dijoint volume and an intersection volume which is a comparison of the sister volumes within the left and right collections.
13	compareCollection (PBSCollection left, PBSCollection right)	Either left or right is empty or null	Return other collection
14	compareCollection (PBSCollection left, PBSCollection right)	right is same as left	Intersection will be copy of left collection and disjoint will be empty.

Level 15

PBSGUI

The PBSGUI is how PBS appears to the user. Through this class, he will be able to send and receive bookmark libraries, change configuration options, see the result of comparing collections from the local machine and the server collection, and extract the bookmarks from his local machine. PBSParser handles these requests. It is finally tested here:

Test Cases

7

Case	Method	Input	Expected Output
1	constructor		set up the GUI to look correctly for the user

Acknowledgements

We would like to thank the following people for their hard work and contributions to the writing of the test plan.

Document Authors:

- Title Page: Nikitas Marangos
- Table of Contents: Nikitas Marangos
- Introduction: Nikitas Marangos, Sean Hanrahan, John Wu
- Unit Testing: Nikitas Marangos, Sean Hanrahan
- Integration Testing: Nikitas Marangos, Sean Hanrahan
- Functional/Acceptance Testing: Nikitas Marangos, Sean Hanrahan
- Integration Schedule: Nikitas Marangos, Sean Hanrahan
- Testing Responsibilities: Nikitas Marangos, Sean Hanrahan, John Wu, Adib Contractor, Jon Eisenstein
- Level 1: Sean Hanrahan, Nikitas Marangos
- Level 2: Nikitas Marangos
- Level 3: Nikitas Marangos
- Level 4: Nikitas Marangos
- Level 5: Nikitas Marangos
- Level 6: John Wu, Nikitas Marangos
- Level 7: John Wu, Nikitas Marangos
- Level 8: Sean Hanrahan, Nikitas Marangos
- Level 9: Sean Hanrahan, Nikitas Marangos
- Level 10: Jon Eisenstein, Nikitas Marangos
- Level 11: Jon Eisenstein, Nikitas Marangos
- Level 12: Nikitas Marangos, Jon Eisenstein
- Level 13: Jon Eisenstein, Nikitas Marangos
- Level 14: Adib Contractor, Nikitas Marangos
- Level 15: Adib Contractor, Nikitas Marangos
- Acknowledgements: Nikitas Marangos, Sean Hanrahan, John Wu

Document Editors:

- Nikitas Marangos

Expert References:

- Alex Borgida, John Wilkes Booth